10/511775

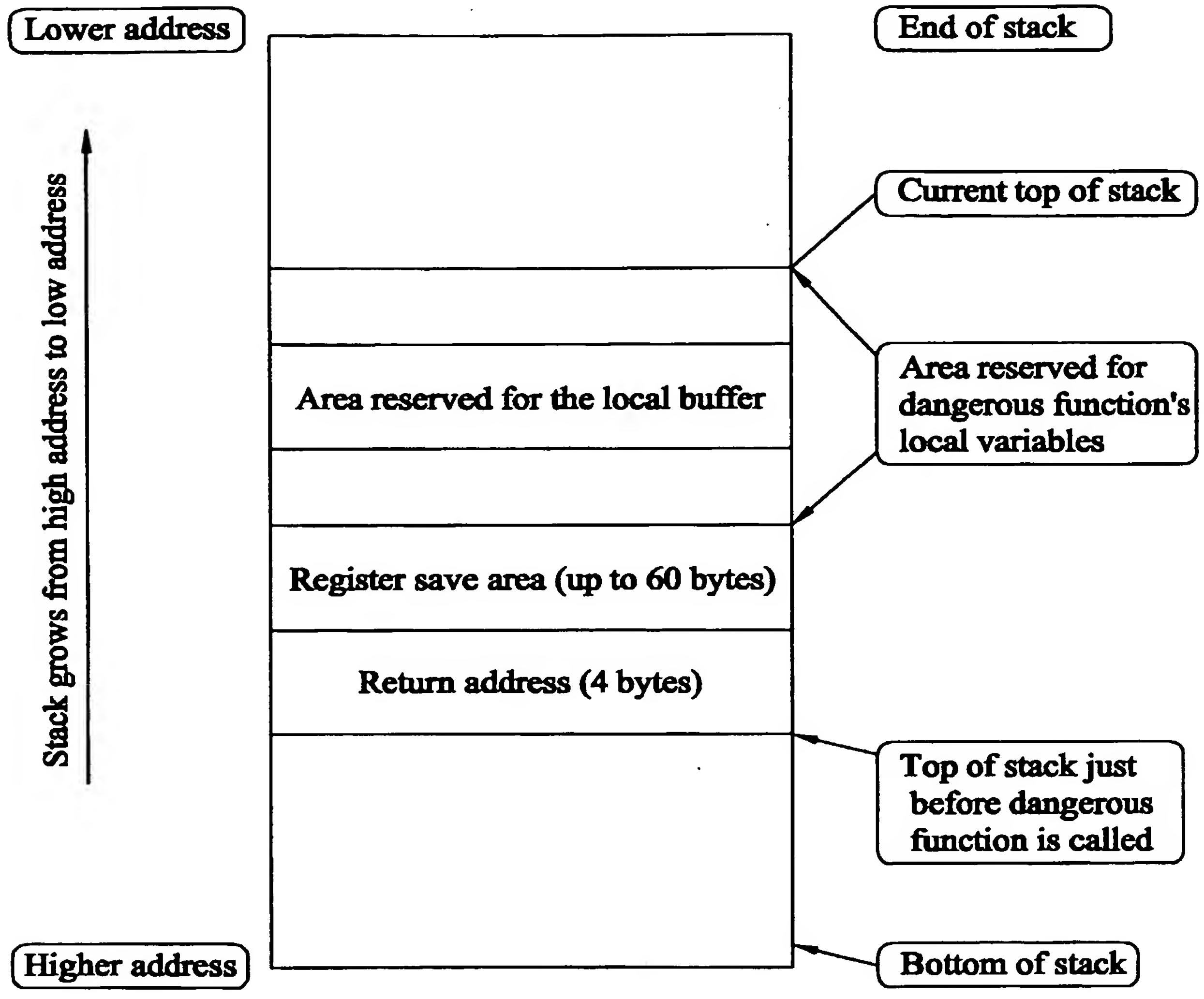
Lower address
End of stack
Stack grows from high address to low address
Current top of stack
Area reserved for the local buffer
Area reserved for dangerous function's local variables
Register save area (up to 60 bytes)
Return address (4 bytes)
Top of stack just before dangerous function is called
Higher address
Bottom of stack

FIG. 1

The stack at the beginning of the call to dangerous function.

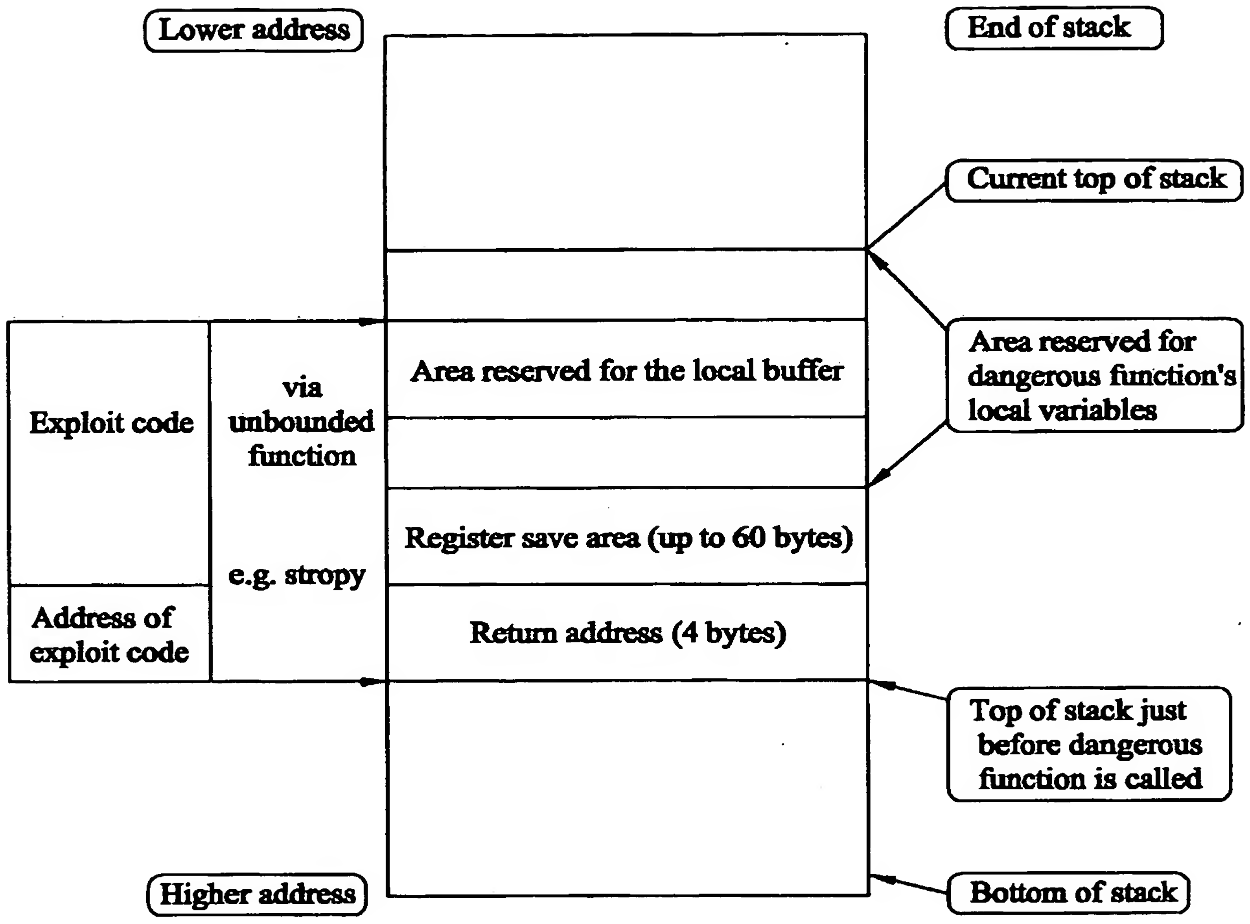
Lower address
End of stack
Current top of stack
Area reserved for dangerous function's local variables
Exploit code
via unbounded function
e.g. stropy
Area reserved for the local buffer
Register save area (up to 60 bytes)
Address of exploit code
Return address (4 bytes)
Top of stack just before dangerous function is called
Higher address
Bottom of stack

FIG. 2

The stack at the point of the unboumded function call.

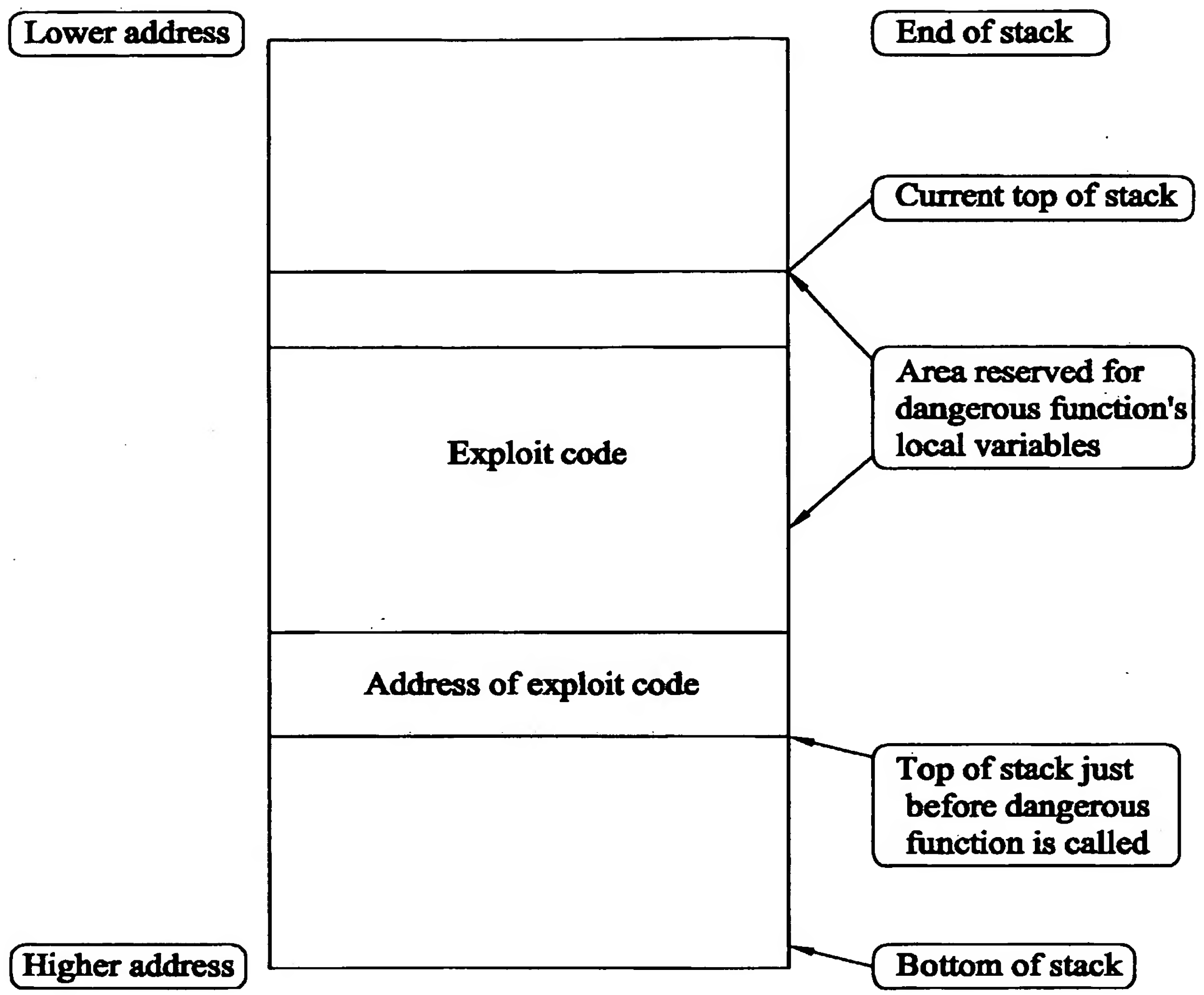

FIG. 1

The stack after the unbounded function call.

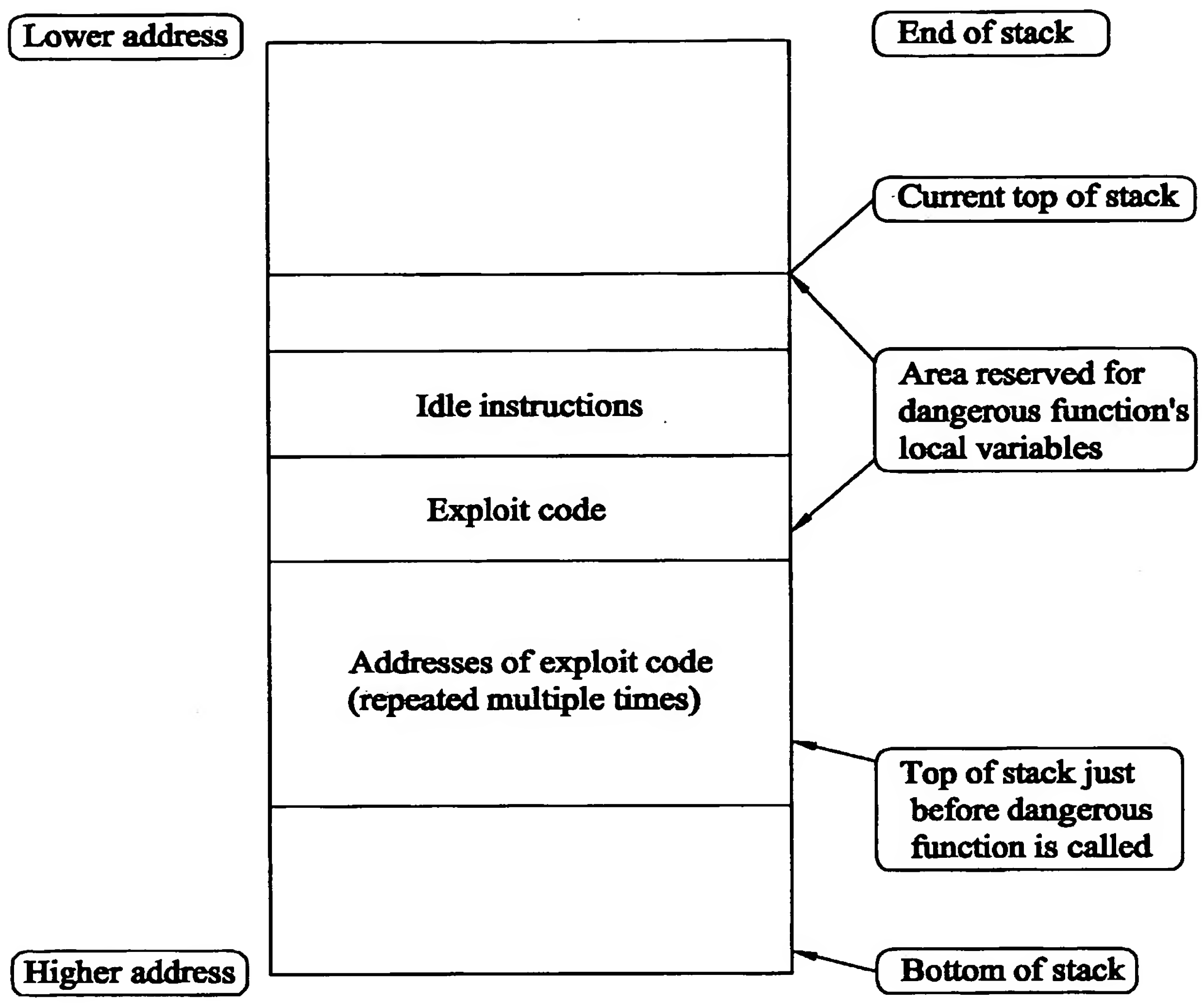
Lower address
End of stack
Current top of stack
Idle instructions
Area reserved for dangerous function's local variables
Exploit code
Addresses of exploit code (repeated multiple times)
Top of stack just before dangerous function is called
Higher address
Bottom of stack

FIG. 4

Revised diagram of the stack after the unbounded function call, incorporating idle sequence and multiple return addresses.

```
  Oxeb,Oxif                  jmp  Ox1f                  jump to call
  Ox5e                       popl   %esi                pop address of string into %esi
  Ox89,Ox76,Ox08             movl  %esi,Ox8(%esi)       place address of string
  Ox31,Oxc0                  xorl   %eax,%eax           generate null long in %eax
  Ox88,Ox46,Ox07             movb  %eax,Ox7(%esi)       terminate string
  Ox89,Ox46,Ox07             movl  %eax,Oxc(%esi)       place null long
  Oxb0,Ox0b                  movb  $Oxb,%al             set system call number
  Ox89,Oxf3                  movl  %esi,%ebx            move address into %ebx
  Ox8d,Ox4e,Ox08             leal    Ox8(%esi),%ecx     load address of address
  Ox8d,Ox56,Ox0c             leal    Oxc(%esi),%edx     load address of null long
  Oxcd,Ox80                  int  $Ox80                 jump to kernel mode
* Ox31,Oxdb                  xorl   %ebx,%ebx           generate null long in %ebx
* Ox89,Oxd8                  movl  %ebx,%eax            move null long into %eax
* Ox40                       inc  %eax                  increment %eax
* Oxcd,Ox80                  int  %Ox80                 jump to kernel mode
  xe8,Oxdc,Oxff,Oxff,Oxff    call  -Ox24                call pop instructionn
  /bin/sh                    .string  "/bin/sh"         shell string
```

FIG. 5

```
% idle/3 - Predicate representing an idle instructions, consisting of
% opcode of instruction in hexadecimal, assembler mnemonic for
% instruction, unique ID of instruction
idle (Ox90, 'nop',0).
idle (Oxfc, 'cld',1).
idle (Oxf9, 'stc',2).
idle (Oxf5, 'cmc',3).
idle (Oxf8, 'clc',4).
idle (Ox99, 'cltd',5).
idle (Ox9b, 'fwait',6).

% idle_sequence/2 - Find the maximum number of consecutive idles
% in the list of bytes
idle_sequence (Bytes, MaxSequence) :-  sequence (Bytes, MaxSequence, 0,0).

sequence [], Max,_,Max).
sequence ([Byte/Rest], Final,Current,Max) :- idle (Byte,_,_ ),
                                             plus (1, Current, NewCurrent),
                                             greater (NewCurrent, Max, NewMax),
                                             sequence (Rest, Final,NewCurrent,NewMax).

sequence ([Byte/Rest], Final,Current,Max) :- not (idle (Byte,_,_ ) ),
                                             sequence (Rest, Final, 0, Max).

% command/2 - Predicate representing a command, consisting of
% name of command and unique ID of command
command (['/', 'b', 'i', 'n', '/', 's', 'h'], 0).
command (['/', 'b', 'i', 'n', '/', 'b', 'a', 's', 'h'], 1).
command (['/', 'b', 'i', 'n', '/', 'c', 's', 'h'], 2).
command (['/', 'b', 'i', 'n', '/','t', 'c', 's', 'h'], 3).
command (['/', 'b', 'i', 'n', '/', 'a', 's', 'h'], 4).
command (['/', 'b', 'i', 'n', '/', 'b', 's', 'h'], 5).

%command_command/1 - Is it true if the list of bytes contains a command
contains_command (Bytes) :- command (Command, _),
                                     concat (_, B2, Bytes),
                                     concat (Command, _, B2,).

% utility predicates
greater (A, B, A) :- A > B.
greater (A, B, B) :- B =< B.
plus (A, B, C) :- C is A + B.
concat ([], L, L).
concat ([X/L1], L2, [X/L3]) :- concat (L1, L2, L3).
```

FIG. 6

Predicates from the Knowledge Base.

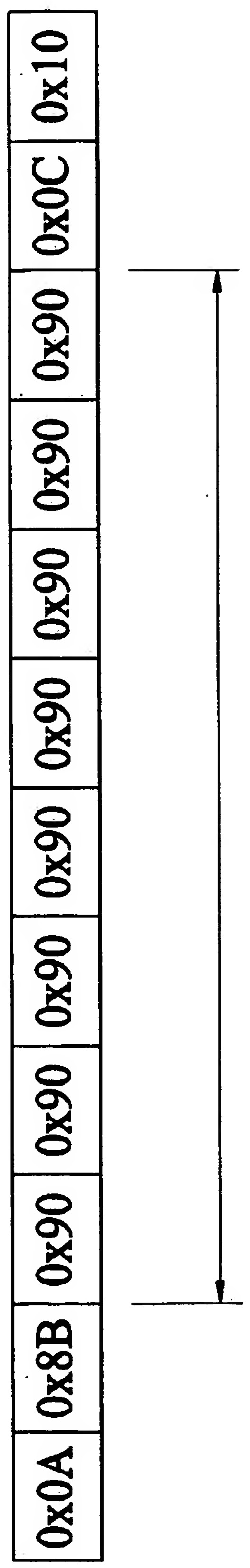

FIG. 7

NOP sequence detected by a typical IDS and the Prolog Knowledge Base.

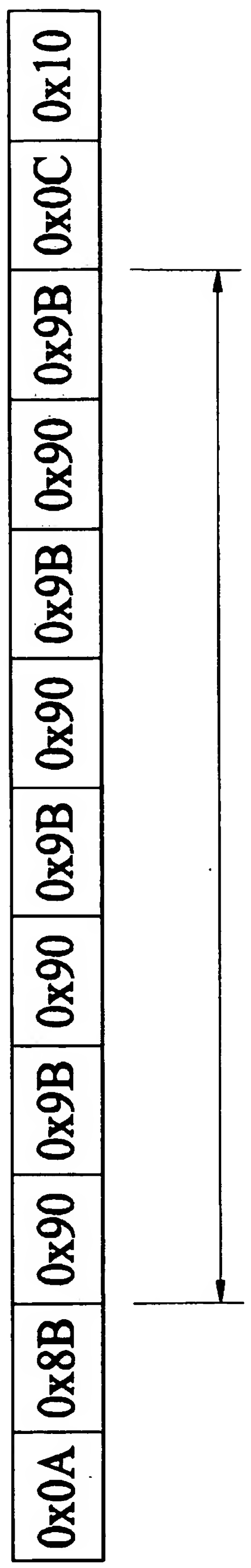

FIG. 8

NOP & FWAIT sequence detected by the Prolog Knowledge Base. A typical IDS generates a false negative.

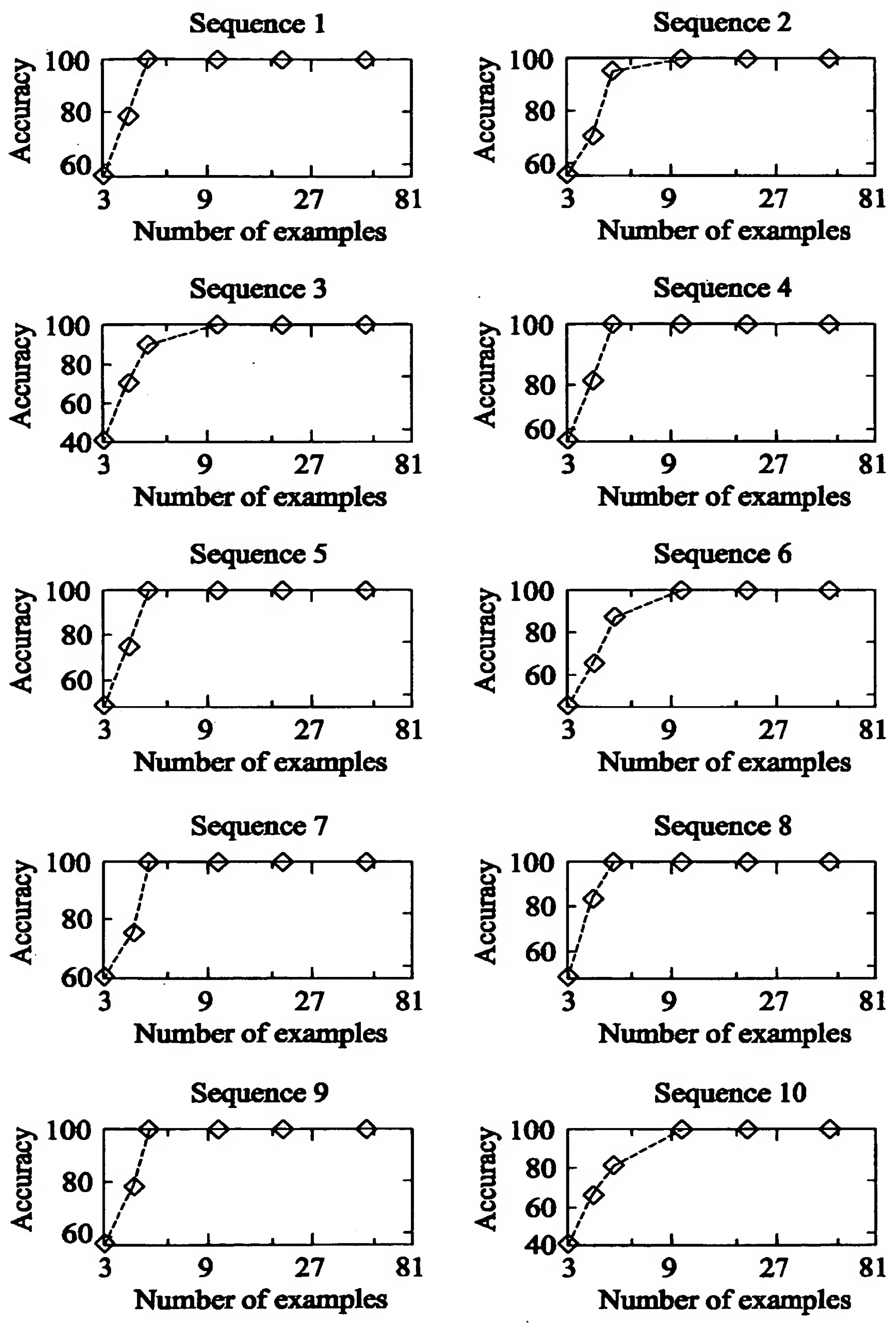

FIG. 9

Experimental results for each sequence.

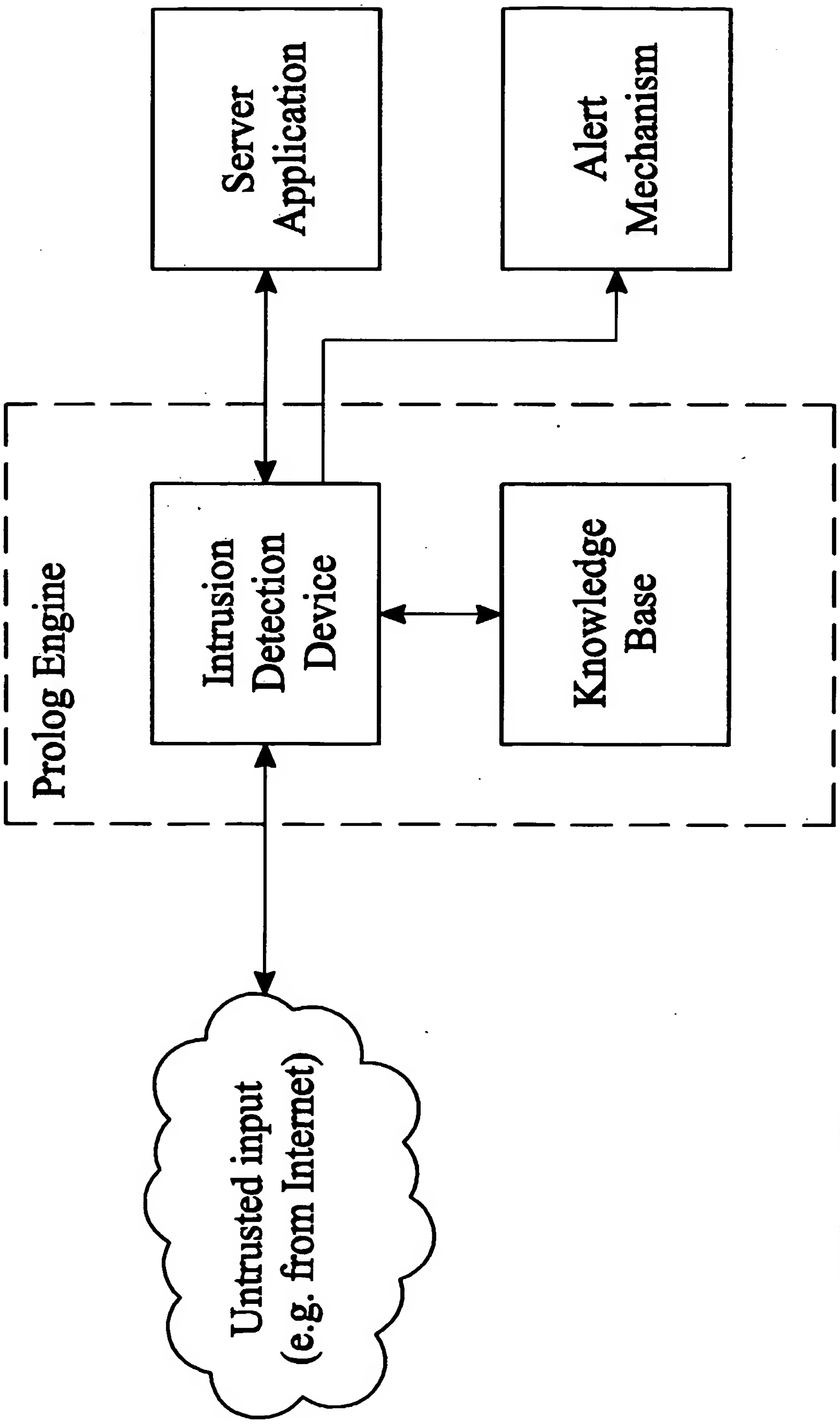
Server Application
Alert Mechanism
Prolog Engine
Intrusion Detection Device
Knowledge Base
Untrusted input (e.g. from Internet)

FIG. 10